Assignment 1, B461, 9/2/20
Andrew Jedlicka

Problem 1.2

1.
What was done: changed primary key in employee to be ename instead of id and removed all foreign keys*
What was learned: Changing the primary key to the name, or any value, keeps duplicate entries of that key from being entered, even if every other value in the entry is different.
Examples:
sql:
```
INSERT INTO employee VALUES
   (1001,'Jean','Bloomington','Apple',60000),
   (1001,'Andy','Indianapolis','Google',100000);
SELECT * FROM employee;
```
outputs:
```
  id  | ename |    city     | cname | salary
------+-------+-------------+-------+--------
 1001 | Jean  | Bloomington | Apple |  60000
 1001 | Andy  | Bloomington | Apple |  60000
(2 rows)
```
sql:
```
INSERT INTO employee VALUES (1001,'Jean','Bloomington','Apple',60000);
INSERT INTO employee VALUES (1002,'Jean','Indianapolis','Google',65000);
SELECT * FROM employee;
```
outputs:
error: "DETAIL: Key (ename)=(Jean) already exists."
```
    id  | ename |    city     | cname | salary
  ------+-------+-------------+-------+--------
 1001 | Jean  | Bloomington | Apple |  60000
(1 row)
```
sql:
```
INSERT INTO employee VALUES
  (1001,'Jean','Bloomington','Apple',60000),
  (1002,'Jean','Indianapolis','Google',65000);
SELECT * FROM employee;
```
outputs
```
 id | ename | city | cname | salary
----+-------+------+-------+--------
(0 rows)
```

2.

What was done: changed primary key in employee to (id, ename) instead of id and removed all foreign keys*

What was learned: Changing the primary key to the (id, ename) keeps duplicate entries of both id and name from being entered. Only if an insert had both the same name and id would the INSERT throw an error.

Examples:

sql:

```
INSERT INTO employee VALUES
  (1001,'James','Bloomington','Apple',60000),
  (1002,'Jen','Indianapolis','Google',65000),
  (1003,'Jean','Indianapolis','Google',65000);
SELECT * FROM employee;
```

outputs:

```
 id  |  ename   |     city     | cname  | salary
------+----------+--------------+--------+--------
 1001 | Jen      | Bloomington  | Apple  |  60000
 1002 | Jen      | Indianapolis | Google |  65000
 1002 | Jennfier | Indianapolis | Google |  65000
```

sql:

```
INSERT INTO employee VALUES
  (1001,'Jen','Bloomington','Apple',60000),
  (1002,'Jen','Indianapolis','Google',65000),
  (1002,'Jennfier','Indianapolis','Google',65000),
  (1001,'Jen','Indianapolis','Google',65000);
SELECT * FROM employee;
```

outputs:

error: DETAIL: Key (id, ename)=(1001, Jen) already exists.

```
 id | ename | city | cname | salary
----+-------+------+-------+--------
(0 rows)
```

3.

What was done: removed FOREIGN KEY (cname, city) REFERENCES company (cname, city)

What was learned: In the case of removing FOREIGN KEY (cname, city) REFERENCES company (cname, city), the city and cname are free to be anything, whereas with the foreign key, the company and city would be regulated to a company and city combination registered in the company table.

Examples:

sql:

```
INSERT INTO employee VALUES
  (1001,'James','Bloomington','Apple',60000),
  (1002,'Jen','Timbuktu','Google',65000),
  (1003,'Jean','Indianapolis','Google',65000);
```

SELECT * FROM employee;
ouputs:

```
 id  | ename |     city     | cname  | salary
------+-------+--------------+--------+--------
 1001 | James | Bloomington  | Apple  |  60000
 1002 | Jen   | Timbuktu     | Google |  65000
 1003 | Jean  | Indianapolis | Google |  65000
(3 rows)
```

4.
What was done: removed FOREIGN KEY (mid) REFERENCES employee (id), FOREIGN KEY (eid) REFERENCES employee (id)
What was learned: Without FOREIGN KEY (mid) REFERENCES employee (id), FOREIGN KEY (eid) REFERENCES employee (id), there are no regulations on what the mid or eid should be, making these values practically meaningless in regards to the ids of employees.
Examples:
sql:

```
INSERT INTO company VALUES
  ('Apple', 'Bloomington'),
  ('Apple', 'Indianapolis'),
  ('Amazon', 'Bloomington');
INSERT INTO employee VALUES
  (1001,'Jean','Bloomington','Apple',60000),
  (1002,'Vidya', 'Indianapolis', 'Apple', 45000),
  (1003,'Anna', 'Bloomington', 'Amazon', 55000);
INSERT INTO manages VALUES
  (1001, 1002),
  (5, 700000),
  (90000000, -5);
SELECT * FROM employee;
SELECT * FROM manages;
```

outputs:

```
 id  | ename |     city     | cname  | salary
------+-------+--------------+--------+--------
 1001 | Jean  | Bloomington  | Apple  |  60000
 1002 | Vidya | Indianapolis | Apple  |  45000
 1003 | Anna  | Bloomington  | Amazon |  55000
(3 rows)
  mid    |  eid
----------+--------
    1001 |   1002
       5 | 700000
 90000000 |     -5
(3 rows)
```

5.
What was done: tried to delete company table, with original foreign keys and primary keys
What was learned: While there's an employee working at a given company, you cannot delete a company from company table because of the foreign key reference without using CASCADE.
Examples:
sql:

```
INSERT INTO company VALUES
  ('Apple', 'Bloomington'),
  ('Apple', 'Indianapolis'),
  ('Amazon', 'Bloomington');
INSERT INTO employee VALUES
  (1001,'Jean','Bloomington','Apple',60000),
  (1002,'Vidya', 'Indianapolis', 'Apple', 45000),
  (1003,'Anna', 'Bloomington', 'Amazon', 55000);
DELETE FROM company c WHERE c.cname = 'Apple' AND c.city = 'Bloomington';
SELECT * FROM employee;
SELECT * FROM company;
DROP company;
```

outputs:
error: DETAIL: Key (cname, city)=(Apple, Bloomington) is still referenced from table "employee".

```
 id  | ename |     city     | cname  | salary
------+-------+--------------+--------+--------
 1001 | Jean  | Bloomington  | Apple  |  60000
 1002 | Vidya | Indianapolis | Apple  |  45000
 1003 | Anna  | Bloomington  | Amazon |  55000
(3 rows)
 cname  |     city
--------+--------------
 Apple  | Bloomington
 Apple  | Indianapolis
 Amazon | Bloomington
(3 rows)
```

6.
What was done: removed all foreign keys and primary keys altogether
What was learned: in this case, no linking throughout the database was made and no connections were set up, resulting in completely free entry of values throughout and each table acting separately from the others.
Examples:
sql:

```
INSERT INTO company VALUES
  ('Apple', 'Bloomington'),
  ('Apple', 'Indianapolis'),
```

```
  ('Amazon', 'Bloomington');
INSERT INTO employee VALUES
  (1001,'Jean', 'Timbuktu','Apple',60000),
  (1002,'Vidya', 'Indianapolis', 'Apple', 45000),
  (1003,'Anna', 'Bloomington', 'Amazon', 55000);
INSERT INTO manages VALUES
  (1001, 1002),
  (1002, 2001);
SELECT * FROM employee;
SELECT * FROM company;
SELECT * FROM manages;
DROP company CASCADE;
```

outputs:

```
company
 id  | ename |     city     | cname  | salary
------+-------+--------------+--------+--------
 1001 | Jean  | Timbuktu  | Apple  |  60000
 1002 | Vidya | Indianapolis | Apple  |  45000
 1003 | Anna  | Bloomington  | Amazon |  55000
(3 rows)
 cname  |     city
--------+--------------
 Apple  | Bloomington
 Apple  | Indianapolis
 Amazon | Bloomington
(3 rows)
 mid  | eid
------+------
 1001 | 1002
 1002 | 2001
(2 rows)
```

* removal of all foreign keys is to see how table handles other keys as primary key without getting errors from foreign keys